L10: Entry 8 of 9 File: USPT Jan 3, 1995

DOCUMENT-IDENTIFIER: US 5379389 A
TITLE: Method for transmitting commands excluded from a predefined command set

Brief Summary Text (5):
As illustrated in FIG. 1, a typical computer system of any size includes a central processor 10 which executes, in broad terms, two types of software. The type of software which computer users are most familiar with is generally termed application software 12. Examples of application software are word processors, accounting programs, database managers, communication programs, etc. Execution of application software requires an operating system 14. There any many modules included within an operating system. One module, driver software 16, is of primary interest although there are other operating system modules 18 which are also included in the operating system 14.

Brief Summary Text (12):
The above objects are attained by providing a method of processing instructions, each having a command area and a data area, the instructions including first and second sets of commands, the first sets of commands identified by codes in the command area, the method comprising the steps of identifying one of the second set of commands in dependence upon codes in the data area of one of the instructions and executing the one of the second set of commands identified. In one embodiment of the invention, the second set of commands are identified by detecting a predetermined series of codes in the data area indicating that one of the second set of commands is included in the data area of the instruction. Preferably, detection of the second series of codes is performed only if a predetermined command and a predetermined address are detected in the command area of the instruction.

Detailed Description Text (7):
One method of meeting both criteria is to use the contents of the command area 24 to partially identify the existence of a command in the data area 26. An example is illustrated in the flowchart of FIG. 3. First, the function code field is checked 40 to determine whether a specific command in the first (predetermined) set of commands, such as a write command, is present. If so, the address (LBN) is checked 42 to determine whether a specific logical block number is being addressed. Where the peripheral equipment 20 is a disk drive, such as an opto/magnetic disk drive which can access a large amount of memory, e.g., approximately 600 megabytes, these two steps alone will eliminate almost all commands in the first command set very quickly. As a result, the commands will be processed 44 as normal commands.

Detailed Description Text (14):
In all three of the above examples, the present invention uses the application software 12 executed by the processor 10 as command generation means for storing command identification codes and one of the second set of commands in the data area 26 of one of the instructions, where the first set of commands are those defined by the computer system manufacturer (IBM, DEC or Sun) and the second set of commands are additional commands outside the first set. Also, all three examples include command identification means for identifying existence of the command identification codes in the data area of the one of the instructions and command execution means for executing the one of the second set of commands when the command identification codes are identified by the command identification means.

Other Reference Publication (4):
IBM Technical Disclosure Bulletin, vol. 26, No. 11, Apr. 1984, "Transparent Mode In An I/O Controller" by B. L. Beukema, R. C. Booth and E. C. Grazier, pp. 5956-5959.

CLAIMS:

1. A method of processing instructions in a data processing apparatus, each instruction having a command area including an address area, and a data area, the instructions including first and second sets of commands, the first set of commands identified by codes in the command area, said method comprising the steps of:

(a) detecting one of the second set of commands in the data area of one of the instructions without reference to the command area of the instruction;

(b) identifying and executing the one of the second set of commands, detected in step (a), in the data processing apparatus without the codes in the command area having any effect on said executing; and

(c) executing one of the first set of commands as identified by the command area when an instruction is received and step (a) does not detect any of the second set of commands.

7. A method as recited in claim 4, wherein said executing in step (b) comprises the steps of:

(b1) reading a specific series of codes representing the one of the second set of commands from the data area following the predetermined series of codes indicating that any one of the second set of commands is included in the data area of the one of the instructions; and

(b2) executing the one of the second set of commands read in step (b1).

9. A system for transmitting instructions, having a command area including an address area, and a data area, from an originating program to a receiving program, the instructions including first and second sets of commands, the first set of commands identified by codes in the command area, said system comprising:

command generation means for storing command identification codes and one of the second set of commands in the data area of one of the instructions;

command identification means for determining existence of the command identification codes in the data area of the one of the instructions; and

command execution means for executing the one of the second set of commands when the command identification codes are determined by said command identification means to exist in the data area in the one of the instructions and for executing one of the first set of commands in the command area when the command identification codes are not determined to exist in the data area of the one of the instructions.

Previous Doc Next Doc Go to Doc#

First Hit Fwd Refs Previous Doc Next Doc Go to Doc#

□ Generate Collection Print

L10: Entry 4 of 9 File: USPT Apr 6, 1999

DOCUMENT-IDENTIFIER: US 5892939 A
TITLE: Emulator for visual display object files and method of operation thereof

Detailed Description Text (2):
Before describing the system and method of the present invention, it will be helpful in understanding a system environment in which the present invention may be used. Accordingly, referring initially to FIG. 1, illustrated is a block diagram of a process control system 100 in which the system and method of the present invention may be found. Process control system 100 includes a plant control network 110 coupled to a plurality of process controllers 120, 125 via a network interface module ("NIM") 140 and highway gateway ("HG") 145 over a universal control network ("UCN") 130 and data highway 135, respectively. Additional process controllers can be operatively connected to plant control network 110, if so desired. Process controllers 120, 125 interface analog input and output signals and digital input and output signals ("A/I," "A/O," "D/I" and "D/O", respectively) to process control system 100 from a variety of field devices (not shown) including valves, pressure switches, pressure gauges, thermocouples or the like.

Detailed Description Text (8):
Another type of physical module included in plant control network 110 is HM 170 that provides mass data storage capability. Each HM 170 includes at least one conventional disk mass-storage device that provides a large volume of storage capability for binary data. The types of data stored by such mass storage device are typically trend histories or data from which such trends can be determined, data that constitute or form displays, copies of programs for the units of process controllers 120, 125 for the modules (e.g., US 150, AM 160) or for units of the modules of plant control network 110.

Detailed Description Text (9):
Another type of module incorporated into plant control network 110 is AM 160. AM 160 provides additional data processing capability in support of the process control functions performed by process controllers 120, 125, such as data acquisition, alarming, batch history collection and providing continuous control computational facilities when needed. The data processing capability of AM 160 is provided by its module processor and module memory (not shown).

Detailed Description Text (10):
CM 180 may use standard or common units of all physical modules to permit a medium-to-large scale, general-purpose data processing system to communicate with other modules (e.g., US 150, AM 160) of plant control network 110 and the units of such modules over LCN 190 and the units of process controllers 120, 125 via NIM 140 or HG 145, respectively. Data processing systems of CM 180 are used to provide supervision, optimization, generalized user program preparation and execution of such programs in higher-level program languages. Typically, the data processing systems of CM 180 have the capability of communicating with other such systems by a communication processor and communication lines, as is well known in the art. CM 180 may include one of several kinds of computers and operating systems. Honeywell DPS-6 computers, for instance, may be employed in CM 180.

Detailed Description Text (45):
Turning to FIG. 4C, illustrated is a high-level block diagram of exemplary processing circuitry (generally designated 310) that may suitably be associated with a computer, such as PC 305, to provide a non-native environment within which the present invention may suitably be implemented and operated. Processing circuitry 310 illustratively includes a processor 470, a conventional random access memory ("RAM") 475, bus controller circuitry 480, a conventional read-only memory ("ROM") 485, a conventional video random access memory ("VRAM") 490 and a set of peripheral ports 495. An exemplary host bus 497 is shown and is suitably operative to associate processor

470,. RAM 475 and bus controller circuitry 480. An exemplary input/output ("I/O") bus 498 is shown and is operative to associate bus controller circuitry 480, ROM 485, VRAM 490 and the set of peripheral ports 495. The set of peripheral ports 495 may suitably couple I/O bus 498 to any one or more of a plurality of conventional suitably arranged peripheral devices for communication therewith. Included among the set of peripheral ports 495 may suitably be one or more serial or parallel ports, such as a conventional PCI slot. In point of fact, a preferred embodiment discussed next uses the PCI slot to associate computer 305 with the process control system through application specific circuitry, LCNP, disclosed in U.S. patent application Ser. Nos. 08/727,724 and 08/727,725 which have been previously incorporated herein by reference for all purposes.

Detailed Description Text (46):
Bus controller circuitry 480 provides suitable means by which host bus 497 and I/O bus 498 may be associated, thereby providing a path and management for communication therebetween. Each of the illustrated buses 497 and 498 requires a drive current to carry signals thereon. The illustrative circuit accordingly operates in conjunction with a conventional system controller (not shown) that supplies the required drive current. Of course, the illustrative circuitry may also suitably be implemented having only a single bus or more than three buses.

Detailed Description Text (73):
WINDOWS.RTM. NT.RTM. starts a software PCI driver early in startup, based on WINDOWS.RTM. NT.RTM. Registry entries. During the WINDOWS.RTM. NT.RTM. startup sequence, a WINDOWS.RTM. NT.RTM. service called "Emulator Service" is automatically started. The main program of this service first initializes its internal database to "empty" and then reads configuration information indicating which emulators are needed.

Detailed Description Text (78):
Draw commands are stored in buffers of "display list" instructions being aggregated for transmission to PDG. They are handed to the PDG via a buffer associated with an I/O control Buffer ("IOCB"), whose processing is initiated with a START COMMAND to the PDG board.

Detailed Description Text (79):
On the WINDOWS.RTM. NT.RTM. side, the issuance of the START COMMAND generates an interrupt to the PCI driver. It determines the address written (command register of the associated slot, in this case), and reads the command register contents (i.e., the command). Detecting that this is a command register write, it may determine the address of the first IOCB. It then determines which CBR threads are interested in this command, and "completes" their outstanding read requests.

Detailed Description Text (84):
Writes from the TDC are issued at a "logical sector" on a designated device. The write request from the TDC results in a SCSI IOCB with a write command being built, followed by the issuance of a START COMMAND to the SCSI slot. The write generates an interrupt to the WINDOWS.RTM. NT.RTM. system. The WINDOWS.RTM. NT.RTM. system responds to the interrupt, notes the START COMMAND, and indicates that the command is accepted.

Detailed Description Text (89):
Note that according to the illustrated embodiment, multiple contiguous 256 byte blocks may be written in response to a single write command, the number of sectors specified in the associated ICOB. Further, in alternate embodiments, the 256 byte value may suitably be altered.

Detailed Description Text (91):
A TDC Read command and a Write command are very similar, the differences however are: (1) the buffer area for the read need not be transferred over to the WINDOWS.RTM. NT.RTM. side--instead, data read from the WINDOWS.RTM. NT.RTM. disk file is transferred back to TDC memory; and (2) if the Read is requested from a sector that is beyond the current length of the file (but is within the legal preconfigured maximum length of the disk), then a dummy record full of all zeroes is returned, but the file is not extended to this length.

Detailed Description Text (92):
Note similarly that according to the illustrated embodiment, multiple contiguous 256 byte blocks may be read in response to a single read command, the number of sectors specified in the

associated ICOB. Further, in alternate embodiments, the 256 byte value may again suitably be altered.

Detailed Description Text (102):
There are two databases in the PCI Driver: (1) slot memory cache; and (2) interrupt receiver registry. The slot memory cache database reflects the entirety of slot memory which exists on the LCNP in lieu of actual slot memory which would normally have been supplied by the LCNP boards. This database is read by and written to WINDOWS.RTM. NT.RTM. applications and (indirectly) written to by TDC (via unsolicited target access). The interrupt receiver registry database records which WINDOWS.RTM. NT.RTM. tasks are interested in being notified of writes to particular slot addresses.

Detailed Description Text (106):
CBR includes a database of supported emulators and corresponding slot number on the PCI interface (read in from the registry).

Detailed Description Text (111):
Not all commands sent by the TDC MMI are drawing commands. Several commands relate to text input port ("TIP") processing. These commands add complexity to the video emulator. In order to correctly handle TIP processing commands, the video emulator must access databases kept in TDC module bus memory and support simple text editing capabilities.

Detailed Description Text (115):
Four databases may be used to perform this blending procedure, namely, (1) text image, a monochrome bitmap of all text currently shown on the screen; (2) graphics image, a monochrome bitmap of all graphics currently shown on the screen; (3) attribute matrix, an n by m matrix of attribute values, wherein n is the number of characters to fill the width of the screen and m is the number of characters to fill the height of the screen--an attribute value indicates the color, intensity, and blink attributes ascribed to the bits composing the character cell and this value additionally includes a priority which indicates how to blend the graphics relative to the text; and (4) blended image, a 16 color bitmap describing the actual image displayed on the screen, this image is required because blending directly to the screen results in unacceptable screen flickering.

Detailed Description Text (116):
There may also be three TDC module bus memory resident databases to support TIP processing commands and cursor shape management, namely, (1) TIP context array, that describes the location of TIP areas on the screen--used to correctly place text on the screen during TIP processing and manage the cursor shape; (2) screen image table, that includes ASCII values of text currently appearing on the screen; and (3) PICK area array, that describes the locations and other information of the PICK areas on the screen.

Detailed Description Text (117):
At the bottom, the video emulator uses the unsolicited target access interfaces at the top of CBR--the solicited target access interface of CBR are enlisted to read and write the module bus memory resident databases. At the top, the video emulator is a write-only application, providing no external interfaces--the interface to the Native Window is provided by the keyboard and pointer emulators.

Detailed Description Text (123):
A single database per keyboard type is required by the keyboard emulator. These databases provide keycode mappings between keycodes generated by WINDOWS.RTM. NT.RTM. keyboard drivers and standard US 150 keycodes. The keyboard emulator also contains databases correlating key sequences to configured actions, at least one describes a Native Window menu and at least one other describes Native Window accelerator key sequences.

Detailed Description Text (151):
Turning now to FIG. 8, illustrated is a conceptual block diagram of an exemplary functional design model of PCI driver according to the principles of the present invention. Exemplary PCI Driver exists within the Device Interface Layer layer of the architecture. It consists of two subsystems, the solicited target access and the unsolicited target access. It contains two databases, the slot memory and the interrupt receiver registry.

Detailed Description Text (154):

Unsolicited target access provides an interface by which its users (environment layer device emulators) can register their interest in writes to I/O space by the TDC. Then, as writes to I/O space occur, interested parties are determined from the registration database and informed of the write event. In addition--and regardless of whether any interested parties are registered for specific writes--unsolicited target access updates the slot memory cache database which is accessed by solicited target access for I/O-space reads.

Previous Doc Next Doc Go to Doc#

Freeform Search

Database:
US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Term: `L9 and (detect$ near command)`

Display: 50 **Documents in Display Format:** FRO **Starting with Number** 1

Generate: ○ **Hit List** ● **Hit Count** ○ **Side by Side** ○ **Image**

Search Clear Interrupt

Search History

DATE: Monday, June 13, 2005 Printable Copy Create Case

DB=USPT; PLUR=YES; OP=OR

Set Name side by side	Query	Hit Count	Set Name result set
L10	L9 and (detect$ near command)	9	L10
L9	L8 and (read near command)	108	L9
L8	L7 and (write near command)	180	L8
L7	L6 and database	781	L7
L6	L5 and controller	5756	L6
L5	stor$ near commands	9428	L5
L4	L3 and (database near interface)	1	L4
L3	L1 and (controller)	18	L3
L2	L1 and (logic near controller)	0	L2
L1	(detect$ near command$) same database	48	L1

END OF SEARCH HISTORY

First Hit Fwd Refs Previous Doc Next Doc Go to Doc#

□ Generate Collection Print

L10: Entry 3 of 9 File: USPT Apr 4, 2000

DOCUMENT-IDENTIFIER: US 6047334 A

**** See image for Certificate of Correction ****

TITLE: System for delaying dequeue of commands received prior to fence command until commands received before fence command are ordered for execution in a fixed sequence

Drawing Description Text (4):
FIG. 2 illustrates a memory controller according to one embodiment of the present invention.

Drawing Description Text (6):
FIG. 4 depicts write and read command queues according to one embodiment of the present invention after receiving a first sequence of commands.

Drawing Description Text (7):
FIG. 5 depicts the write and read command queues of FIG. 4 after receiving a second sequence of commands.

Drawing Description Text (8):
FIG. 6 depicts the write and read command queues of FIG. 5 after receiving a third sequence of commands.

Drawing Description Text (9):
FIG. 7 depicts the write and read command queues of FIG. 6 after a first set of commands have been dequeued.

Drawing Description Text (10):
FIG. 8 depicts write and read command queues according to one embodiment of the present invention having a synchronization value and a fenced memory access command at their respective heads.

Drawing Description Text (11):
FIG. 9 depicts write and read command queues according to one embodiment of the present invention having a non-fenced write command and a synchronization value at their respective heads.

Drawing Description Text (12):
FIG. 10 depicts write and read command queues according to one embodiment of the present invention having a fenced write command and a synchronization value at their respective heads.

Drawing Description Text (13):
FIG. 11 depicts the structure of an entry in a read command queue according to one embodiment of the present invention.

Detailed Description Text (5):
Although texture maps allow 3D scenes to be rendered using far less storage than a database of pre-generated images, texture maps can still require considerable storage space. Also, more detailed scenes typically require more texture maps. It is not uncommon for a large, detailed video game to require as much as 40 megabytes (MB) of storage for texture maps alone.

Detailed Description Text (6):
Of course, the ability to render 3D scenes in real-time requires that texture maps be rapidly accessible and texture maps have traditionally been stored in a specialized memory that has relatively low-access latency and is local to the graphics controller. Unfortunately, specialized graphics controller memory is expensive, and even high-end computer systems often

do not have a large enough graphics controller memory to store all the texture maps for a given scene-rendering program. Also, since the majority of application programs that are run on general-purpose computers do not require such a large graphics controller memory, the cost of a large graphics controller memory is often not worth the benefit.

Detailed Description Text (7):
One technique for providing low-latency access to large texture maps is addressed in a specification called the "Accelerated Graphics Port Interface Specification Revision 1.0" (hereinafter the "AGP specification"), published Jul. 31, 1996 by Intel.TM. Corporation. The AGP specification describes a data and command path called the accelerated graphics port (AGP) through which a graphics controller may achieve relatively low-latency access to a computer system's main memory (typically DRAM). According to the AGP specification, the graphics controller interfaces directly to the memory controller of the main memory instead of accessing main memory via the system bus. This removes the need for the graphics controller to arbitrate with other bus master devices for control of the system bus and therefore lowers the overall memory access latency.

Detailed Description Text (8):
The AGP specification also describes a relaxed memory access protocol in which read operations and write operations may be reordered with respect to one another in order to optimize data transfer to and from main memory. More specifically for a given sequence of read and write commands issued by the graphics controller and executed by a memory controller, the following rules apply:

Detailed Description Text (9):
1. Read commands may be executed out of order with respect to other read commands so long as the data ultimately returned to the graphics controller is ordered according to the original read command sequence.

Detailed Description Text (10):
2. Write commands may be executed out of order with respect to other write commands except that a write command may not be reordered for execution ahead of another write command that references the same memory address.

Detailed Description Text (11):
3. Read commands may be executed out of order with respect to write commands and write commands may be executed out of order with respect to read commands except that a read command may not be reordered for execution ahead of a write command that references the same memory address.

Detailed Description Text (12):
Although performance benefits are achieved by virtue of the relaxed memory access policy (e.g., by producing higher page hit rates to improve memory access time), it is occasionally necessary to ensure that a given memory access command is not reordered for execution ahead of memory access commands previously issued by the graphics controller. This is referred to as "fencing" the order of command execution and is accomplished by using a fence command.

Detailed Description Text (13):
As discussed above, in prior art devices that implement command reordering, fence commands are typically enqueued for processing in the same manner as other commands and therefore consume storage space in the command queue. This is particularly problematic in the context of the AGP, because the AGP specification permits unlimited fence commands to be sent to the memory controller in succession and yet requires that the memory controller always be able to queue a specified number of memory access commands (i.e., non-fence commands). Even if back to back fence commands were collapsed into a single entry in a memory controller command queue, the size of a command queue required to hold N memory access commands would still be 2.times.N to account for a command sequence in which a fence command follows every memory access command.

Detailed Description Text (14):
According to one embodiment of the present invention, rather than enqueue fence commands in a command queue within a memory controller, a flag is set upon receipt of the fence command. The flag is reset upon receipt of the next memory access command and the memory access command is enqueued together with an indication that the memory access command succeeded the fence command. In this way, the fence is indicated without dedicating an entire queue entry to the

fence command.

Detailed Description Text (15):
In the event that the memory controller contains multiple command queues, a synchronization value is enqueued in each other command queue that is used to enqueue commands that must not be reordered across the fence command. The memory access command that succeeded the fence command is then delayed from being dequeued from the command queue into which it was inserted until the synchronization values are advanced to the respective heads of the other command queues. The memory access command that succeeded the fence command is also delayed from being dequeued until commands received prior to the fence command have been flushed from a reordering domain of the memory controller and placed in a fixed order for execution.

Detailed Description Text (17):
FIG. 1 depicts a computer architecture 12 including a processor 7, memory controller 9, main memory 10, graphics controller 11, local graphics memory 13, display device 15 and I/O devices 17. As shown, the processor 7 is coupled to memory controller 9 via a processor bus 5. The processor issues memory read and write request signals to the memory controller 9 which, in response, writes and reads the indicated locations in main memory 10. The processor also issues I/O write and read signals to memory controller 9 which, in turn, transfers the I/O write and read signals to the I/O devices 17 via system I/O bus 3. The I/O devices 17 may include any addressable devices necessary to support the needs of the computing system. For example, if computer architecture 12 is used to implement a general purpose computer, the I/O devices 17 would typically include input devices such as a keyboard and screen pointing device, mass storage devices such as magnetic and optical disk drives, network connection devices such as a modem and an area network card, and so forth.

Detailed Description Text (18):
As shown in FIG. 1, graphics controller 11 has direct access to local graphics memory 13 and also has access to main memory 10 by way of the accelerated graphics port (AGP) 14 to memory controller 9. Graphics controller 11 typically includes one or more processors to perform graphics computations and to output a video data stream to display device 15. The AGP 14 may also be accessed by processor 7 via processor bus 5 and memory controller 9 to write and read graphics controller 11.

Detailed Description Text (20):
FIG. 2 illustrates the memory controller 9 of FIG. 1 according to one embodiment of the present invention. As shown, the AGP coupled to memory controller 9 includes both an AGP data path and an AGP command path. The AGP data path is coupled to transfer data to and from AGP data buffer 25. The AGP command path is coupled to deliver memory access commands to AGP command interface 21. At least three types of commands are received in the AGP command interface 21: memory read commands, memory write commands and fence commands. Herein, memory read commands and memory write commands are referred to collectively as memory access commands. The expression "executable command" also appears occasionally herein and refers to any command or other value (except a synchronization value) for which a command queue entry is allocated.

Detailed Description Text (22):
FIG. 3 illustrates the AGP command interface 21 and AGP command reordering logic 23 according to one embodiment of the present invention. AGP command interface 21 includes AGP command decode logic 30, a write command queue 31, a read command queue 33 and queue advance logic 35. Memory access commands are received in the AGP command decode logic 30 via the AGP command path. The AGP command decode logic 30 decodes the commands to determine whether they are write commands or read commands and then enters the commands into either the write command queue 31 or the read command queue 33 accordingly. As discussed further below, one purpose for having separate command queues for memory write and read commands is to allow the write and read commands to be reordered relative to one another.

Detailed Description Text (23):
As shown in FIG. 3, the AGP command decode logic asserts a pair of command-enqueue signals ("ENQUEUE WR CMD" and "ENQUEUE RD CMD" in FIG. 3) to queue advance logic 35 indicating that either a write command or a read command should be enqueued. Upon receiving a command to enqueue a write command or a read command, queue advance logic 35 adjusts a queue tail pointer (labeled "Q-TAIL" in FIG. 3) to point to the next entry in the write command queue 31 or the read command queue 33, respectively, and the indicated command is stored therein.

Detailed Description Text (24):
According to the AGP specification, the memory controller (e.g., element 9 of FIG. 2) must be capable of enqueuing a specified number of memory access commands (the specified number being referred to herein as N). Also, to avoid overwriting the memory controller command queue, there may not be more than N outstanding memory access commands issued by the graphics controller (e.g., element 11 of FIG. 1). An outstanding memory access command is one which, from the perspective of the graphics controller (e.g., element 11 of FIG. 1), has not been completed. For example, an outstanding memory read command is a read command for which the graphics controller has not received the requested data and an outstanding memory write command is a write command for which the corresponding data has not yet been transferred from the graphics controller to the memory controller. As discussed below, in one embodiment of the present invention, the memory controller signals the graphics controller to transfer the write data when the corresponding write command is received in the AGP command reordering logic 23 of the memory controller.

Detailed Description Text (25):
The foregoing constraints set forth in the AGP specification have implications for the AGP command interface 21. For example, because there is no restriction on the number of successive write or read commands that may be received via the AGP command path, the write command queue 31 must be capable of holding N write commands and the read command queue 33 must be capable of holding N read commands. Also, because there may be no more than N outstanding memory access commands, it is not possible for one of the write and read command queues (31 and 33) to enqueue a memory access command without the other of the write and read command queues (31 and 33) also having an available entry. This can be seen by the following analysis in which E.sub.Q1 is the number of entries in one of the write and read queues (31 and 33) and E.sub.Q2 is the number of entries in the other write and read queues (31 and 33):

Detailed Description Text (28):
Because E.sub.Q2 must be less than N, and because the write and read command queues (31 and 33) are each at least size N, it follows that the one of the write and read queues (31 and 33) having E.sub.Q2 entries must be capable of enqueuing at least one additional value. As discussed below, this circumstance is exploited in the present invention.

Detailed Description Text (29):
Queue advance logic 35 receives write queue and read queue advance signals from AGP command reordering logic 23 and advances the queue head pointer (labeled "Q-HEAD" in FIG. 3) to point to the next entry in the write command queue 31 and read command queue 33, respectively. The effect of advancing a head pointer in either the read command queue 33 or the write command queue 31 is to dequeue the entry previously pointed at by the head pointer. It will be appreciated that by adjusting queue head and tail pointers to dequeue and enqueue commands in the write and read command queues (31 and 33), the queued commands themselves do not have to be moved from location to location to pass through the queue. Instead, only the head and tail pointers need be adjusted to implement the first-in, first-out (FIFO) operation of the queue. When the head or tail reaches a first or last storage location in the N-sized memory element used to implement the queue, the head or tail is wrapped around to the other end of the N-sized memory element. In other words, according to one embodiment of the present invention, write command queue 31 and read command queue 33 are implemented by way of a ring-like queue in which the final storage location is considered to be logically adjacent to the first storage location. It will be appreciated that other embodiments of the write command queue 31 and the read command queue 33 are possible. For example, once enqueued, memory access commands could be shifted from storage location to storage location until finally reaching a head storage location from which they are output to the AGP command reordering logic 23. In such an implementation, the location of the queue head is static so that the queue head pointer is not required. Other FIFO buffer arrangements may be used to implement the write command queue 31 and read command queue 33 without departing from the spirit and scope of the present invention.

Detailed Description Text (30):
As shown in FIG. 3, AGP command decode logic 30 includes a fence flag 32. Fence flag 32 is a storage element that is set to a first logical state when a fence command is received in the AGP command decode logic 30, and reset to a second logical state when a non-fence command (i.e., an executable command) is received in the AGP command decode logic 30. As discussed

further below, when the fence flag is set, indicating that a fence command was the command most recently received in the AGP command decode logic 30, the next memory access command received in the AGP command decode logic 30 is enqueued in either the write command queue 31 or the read command queue 33 along with an indication that the memory access command succeeded the fence command. According to one embodiment of the present invention, the fence flag is reset to the second logical state upon system initialization.

Detailed Description Text (32):
Write allocation logic 41 is coupled to forward write commands received from the write command queue 31 to write buffer 43. According to one embodiment of the present invention, write allocation logic outputs at least two signals to dequeue logic 37 based on the state of the write buffer 43. If write buffer 43 has storage available to receive a write command, allocation logic asserts a request signal (indicated as "REQ" in FIG. 3) to dequeue logic 37. If write buffer 43 has been completely flushed (i.e., all write commands previously stored therein have been ordered for execution by command arbitration logic 51), allocation logic 41 outputs an empty signal (indicated as "EMPTY" in FIG. 3).

Detailed Description Text (33):
According to one embodiment of the present invention, when a write command is stored in write buffer 43, data retrieval logic (not shown) within AGP command reordering logic 23 signals the graphics controller (e.g., element 11 of FIG. 1) that the data corresponding to the write command is required. Write arbitration logic 47 determines when data corresponding to a given write command has been received and arbitrates among ready write commands to be forwarded to command arbitration logic 51.

Detailed Description Text (34):
As discussed above, write commands can generally be executed out of order with respect to one another except that a write command cannot be reordered for execution ahead of a write command referencing the same memory address. Write buffer 43 is depicted in FIG. 3 in queue format to emphasize this restriction on the otherwise free reordering of write commands. Write arbitration logic 47 includes logic to implement the above described reordering restriction and also to select from among two or more ready write commands based on certain optimization criteria (e.g., page hit optimization). In an alternative embodiment of the present invention, memory write commands may not be reordered relative to one another, and instead may only be reordered relative to memory read commands.

Detailed Description Text (35):
Read allocation logic 39 forwards read commands from read command queue 33 to available storage locations in read buffer 45. According to one embodiment of the present invention, read allocation logic 39 outputs at least two signals to dequeue logic 37: a request signal to indicate that read buffer 45 has one or more available storage locations, and an empty signal to indicate that read buffer 45 has been completely flushed (i.e., all read commands previously stored in read buffer 45 have been ordered for execution by command arbitration logic 51).

Detailed Description Text (36):
As discussed above, there is no restriction on the reordering of read commands relative to one another so that read arbitration logic 49 selects from read commands in read buffer 45 according to optimization criteria (e.g., page hit optimization). In FIG. 5, read buffer 45 is shown in a lateral format to emphasize this unrestricted reordering policy.

Detailed Description Text (37):
Command arbitration logic 51 arbitrates between write commands received from write arbitration logic 47 and read commands received from read arbitration logic 49 according to certain optimization criteria. Command arbitration logic outputs control signals to memory access logic 27 according to the selected write or read command. From the perspective of the AGP command interface 21 and the AGP reordering logic 23, once control signals corresponding to a selected write or read command have been output by the command arbitration logic, the selected write or read command is considered to be retired. After memory access commands have been used to generate control signals to memory access logic 27, they may not be further reordered relative to one another. Such memory access commands are said to have been ordered for execution in a fixed sequence.

Detailed Description Text (38):

FIG. 4 depicts the state of the write and read command queues after the following exemplary sequence of memory access commands have been received and before the commands have been dequeued into the AGP command reordering logic:

Detailed Description Text (40):
As shown in FIG. 4, the write command WR1 has been enqueued in the write command queue 31, and the two read commands, RD1 and RD2, have been enqueued in the read command queue 33. In FIG. 4, the write and read command tail pointers are pointed at storage locations in their respective queues to which the most recent memory access command has been written. Because, at this point, WR1 is the only command enqueued in the write command queue 31, the head and tail pointers for the write command queue 31 point to the same storage location. It will be appreciated that, in an alternative embodiment, the tail pointer could point to the next vacant storage location in the queue instead of the most recently filled location.

Detailed Description Text (41):
FIG. 5 depicts the state of the write and read command queues after the following exemplary sequence of commands is received in the AGP command decode logic 30 and before WR1, RD1 or RD2 have been dequeued:

Detailed Description Text (42):
As described above, when a fence command is received in the AGP command decode logic 30, the fence flag 32 is set. Note that the fence command itself is not enqueued so that unlike prior-art techniques, no queue storage is consumed by the fence command. If a memory access command is received while the fence flag 32 is set, the memory access command is enqueued in the write or read command queue (31, 33) together with an indication that the command succeeded a fence command.

Detailed Description Text (43):
According to one embodiment of the present invention, an additional bit, called a "fence bit", is provided in each storage element within the write and read command queues. An enqueued memory access command having a set fence bit is referred to as a "fenced" memory access command. For example, a read command received in the AGP command decode logic 30 while the fence flag 32 is set is enqueued in the read command queue 33 with a set fence bit and is referred to as a fenced read command. A write command received in the AGP command decode logic 30 when the fence flag is set is likewise enqueued in the write command queue 31 with a set fence bit and is referred to as a fenced write command.

Detailed Description Text (44):
Based on the foregoing discussion, fence flag 32 is set when the fence command is received in the AGP command decode logic 30, and then, when the read command RD3 is received, it is enqueued in the read command queue 33 with a set fence bit (hence the designation "FENCED RD3" in FIG. 5). According to one embodiment of the present invention, in response to receiving the read command while the fence flag is set, a synchronization value is enqueued in the write command queue 31 concurrently or immediately after the fenced read command FENCED RD3 is enqueued in the read command queue 33. Recall from earlier discussion that it is not possible for one of the write and read command queues to enqueue a memory access command without the other of the write and read command queues also having an available entry. Thus, because there is room in read command queue 33 to enqueue FENCED RD3, it follows that there is room in write command queue 31 to enqueue the synchronization value.

Detailed Description Text (45):
In FIG. 5, the enqueued synchronization value is designated "FENCED NOP1". The reason for the terminology "FENCED NOP1" is that, according to one embodiment of the present invention, a synchronization value is an invalid command or no-operation (NOP) indicated by a synchronization bit. As discussed below, the synchronization bit may be provided for by an extra bit in each entry of the write and read command queues (31 and 33).

Detailed Description Text (46):
As described below, the FENCED RD3 and FENCED NOP1 entries in the write and read command queues (31 and 33) define a fence across which command reordering may not occur. This is signified in FIG. 5 by the dashed line labeled "FENCE" extending between the FENCED RD3 and FENCED NOP1 queue entries.

Detailed Description Text (47):
FIG. 6 depicts the state of the write and read command queues after the following exemplary sequence of commands is received in the AGP command decode logic 30 and before the commands received in exemplary command sequence 1 and exemplary command sequence 2 have been dequeued:

Detailed Description Text (48):
It will be appreciated that command dequeuing may occur concurrently with command enqueuing so that it is likely that commands received in at least exemplary command sequence 1 would likely have been dequeued by the time exemplary command sequence 3 is received. Assuming that exemplary sequence 1 commands have not yet been dequeued is nonetheless helpful for understanding the manner in which commands are enqueued in the write and read command queues 31 and 33.

Detailed Description Text (49):
As shown in FIG. 6, read command RD4 is enqueued in the read command queue 33 behind fenced read command FENCED RD3. Likewise write commands WR2, WR3 and WR4 are enqueued in the write command queue 31 behind synchronization value FENCED NOP1. Recall that the fence flag 32 is reset upon receipt of a non-fence command so that receipt of command RD3 in exemplary command sequence 2 caused the fence flag 32 to be reset. For this reason, the fence bit is not set when write command WR2 (the next command received after RD3) is enqueued.

Detailed Description Text (50):
As indicated in exemplary command sequence 3, a fence command follows WR4 so that the fence flag 32 is set when command WR5 is received in the AGP command decode logic 30. Consequently, the fence bit is set when WR5 is enqueued in the write command queue 31 as indicated by the designation "FENCED WR5" in FIG. 6. Also, a synchronization value designated "FENCED NOP2" is enqueued in the read command queue 33 either concurrently with or immediately after the enqueuing of WR5. A dashed line labeled "FENCE" is shown extending between FENCED WR5 and FENCED NOP2 to indicate that commands may not be reordered across the fence.

Detailed Description Text (51):
FIG. 7 depicts the state of the write and read command queues (31 and 33) after read commands RD1 and RD2 have been dequeued. At this point, the fenced read command FENCED RD3 is present at the head of the read command queue 33, but the corresponding synchronization value FENCED NOP1 has not yet advanced to the head of the write command queue 31. According to one embodiment of the present invention, a fenced memory access command cannot be dequeued from either the read command queue 33 or the write command queue 31 unless the following two conditions are satisfied:

Detailed Description Text (52):
1. A synchronization value corresponding the fenced memory access command enqueued in one of the write and read command queues has advanced to the head of the other of the write and read command queues.

Detailed Description Text (55):
Returning to FIG. 7, the first condition set forth above requires that the fenced read command FENCED RD3 be delayed or blocked from being dequeued at least until the write command WR1 is dequeued and the synchronization value FENCED NOP1 is advanced to the head of the write command queue 31. The effectiveness of the synchronization value FENCED NOP1 can now be appreciated. The synchronization value indicates which commands from the write command queue may be dequeued into the AGP command reordering logic (element 23 of FIG. 3) without crossing the fence. This is significant in view of the fact that memory access commands may otherwise be dequeued from the write command queue 31 and the read command queue 33 independently of one another. Also, enqueuing synchronization values to correspond to fenced memory access commands does not require enlargement of read or write command queues.

Detailed Description Text (56):
FIG. 8 depicts the state of the write and read command queues (31 and 33) after the write command WR1 has been dequeued from the write command queue 31. After condition two (set forth above) for dequeuing fenced memory access commands is satisfied, fenced read command FENCED RD3 may be dequeued. As shown in FIG. 8, additional memory access commands WR6, WR7, RD5 and RD6 have been received since the time at which the write and read command queues were in the state depicted in FIG. 7.

Detailed Description Text (57):
According to one embodiment of the present invention, fenced read command FENCED RD3 and synchronization value FENCED NOP1 are dequeued concurrently. However, because the synchronization bit is set in the synchronization value, write buffer allocation logic (element 41 of FIG. 3) in the AGP command reordering logic (element 23 of FIG. 3) does not buffer the synchronization value for execution.

Detailed Description Text (58):
FIG. 9 depicts the state of the write and read command queues 31 and 33 after the synchronization value FENCED NOP1, read commands FENCED RD3 and RD4, and write commands WR2 and WR3 have been dequeued. At this point, the synchronization value FENCED NOP2 appears at the head of the read command queue 33. According to one embodiment of the present invention, the conditions that must be satisfied before a fenced memory access command may be dequeued must also be satisfied before a synchronization value may be dequeued. In other words, regardless of whether a fenced memory access command or the corresponding synchronization value first reaches the head of its respective queue, neither can be dequeued until the other also reaches the head of its queue.

Detailed Description Text (59):
FIG. 10 depicts the state of the write and read command queues 31 and 33 after the fenced write command FENCED WR5 has advanced to the head of the write command queue 31. Once it is determined that the previously dequeued memory access commands have been flushed from the AGP command reordering logic (e.g., element 23 of FIG. 3), the fenced write command FENCED WR5 and the synchronization value FENCED NOP2 may both be dequeued. As indicated by the exemplary enqueued commands WR8, RD7, RD8 and RD9, memory access commands continue to be enqueued in the manner thus described.

Detailed Description Text (60):
FIG. 11 depicts the structure of an entry 103 in read command queue 33 according to one embodiment of the present invention. As shown, read command queue entry 103 includes a command storage area 106 to store a memory read command, a fence bit 104 that is either set or reset to indicate a fenced or non-fenced read command, and a synchronization ("sync") bit 105 to indicate whether the value in command storage 106 is a valid read command or the overall entry is a synchronization value. While the entry 103 is depicted as part of the read command queue 33, it will be appreciated that write command queue 31 may include entries having a similar structure. Also, the exact location of the sync and fence bits (105 and 104) within entry 103 and the exact number of bits allocated to the sync and fence indicators may be changed without departing from the spirit and scope of the present invention. Further, the sync and fence indicators may even be maintained separately from the write and read command queues 31 and 33 so long as the indicators can be associated with entries in the write and read command queues 31 and 33.

Detailed Description Text (61):
FIG. 12 is a diagram depicting in greater detail the signals input to and output from dequeue logic 37. Recall that dequeue logic 37 is the component of the AGP command reordering logic 23 responsible for issuing write and read advance signals to the write and read command queues (elements 31 and 33 of FIG. 3), respectively. The write advance and read advance signals output by dequeue logic 37 serve the purpose discussed above in reference to FIG. 3. The write buffer empty, write command request, read buffer empty and read command request signals (depicted in FIG. 12 as "WR BUFFER EMPTY", "WR CMD REQUEST", "RD BUFFER EMPTY" and "RD CMD REQUEST", respectively) are the EMPTY and REQ signals issued to dequeue logic 37 by the write allocation logic (element 41 of FIG. 3) and the EMPTY and REQ signals issued to dequeue logic 37 by the read allocation logic (element 39 of FIG. 3), respectively, and also serve the purpose discussed above in reference to FIG. 3.

Detailed Description Text (62):
According to one embodiment of the present invention, the write fence bit, write sync bit, read fence bit and read sync bit signals (depicted in FIG. 12 as "WR FENCE BIT", "WR SYNC BIT", "RD FENCE BIT", and "RD SYNC BIT", respectively) input to dequeue logic 37 are received from the fence and sync bits of the entries at the respective heads of the write and read command queues (elements 31 and 33, respectively, of FIG. 3). As discussed below, each of the signals input to the dequeue logic 37 is used in branching steps of a method according to one embodiment of the

present invention.

Detailed Description Text (63):
FIG. 13 depicts a method 100 implemented in dequeue logic 37 according to one embodiment of the present invention. At step 55, signal RD FENCE BIT is examined to determine if a fenced read command is present at the head of the read command queue (element 33 of FIG. 3). If not, then signal WR FENCE BIT is examined at step 57 to determine if a fenced write command is present at the head of the write command queue (element 31 of FIG. 3). If a fenced write command is not present at the head of the write command queue, then the signal RD CMD REQUEST is examined at step 59 to determine whether a request to dequeue a read command is pending from the read allocation logic (element 39 of FIG. 3). If not, then signal WR CMD REQUEST is examined at step 61 to determine whether a request to dequeue a write command is pending from the write allocation logic (element 41 of FIG. 3). If a request to dequeue a write command is not pending, method execution loops back to step 59 to check again for a request to dequeue from the read command queue.

Detailed Description Text (64):
If a request to dequeue a command from the read command queue is detected in step 59, a read command is dequeued at step 63 and method execution loops back to step 55. Similarly, if a request to dequeue a command from the write command queue is detected in step 61, a write command is dequeued at step 65 and method execution loops back to step 55. It will be appreciated that to optimize the execution of method 100, it may be desirable to return to step 57 from step 65 and also to skip the execution of step 57 after looping from step 63 to step 55. These and other optimizations of method 100 are considered to be within the spirit and scope of the present invention.

Detailed Description Text (65):
If a fenced read command is detected at the head of the read command queue in step 55, then the value present at the head of the write command queue is examined in step 67 to determine if it is a synchronization value. As discussed above, this is accomplished according to one embodiment of the present invention by examining the WR SYNC BIT signal input to the dequeue logic (element 37 of FIG. 12). If a synchronization value is not present at the head of the write command queue, then at step 69 the WR CMD REQUEST signal is iteratively examined until it indicates a request to dequeue a value from the write command queue. Then, at step 71, a write command is dequeued from the write command queue and method execution loops back to step 67 to determine if the value advanced to the head of the write command queue is now a synchronization value.

Detailed Description Text (66):
If, at step 67, a synchronization value is detected at the head of the write command queue, then the WR BUFFER EMPTY and RD BUFFER EMPTY signals input to the dequeue logic by the write and read allocation logic, respectively, are iteratively examined until they indicate that all commands previously dequeued from the write and read command queues have been flushed from the AGP command reordering logic (element 23 of FIG. 3). As indicated above, this is one way to determine whether the previously dequeued commands have been ordered for execution in a fixed sequence.

Detailed Description Text (67):
Once the WR BUFFER EMPTY and RD BUFFER EMPTY signals indicate that previously dequeued commands have exited the AGP command reordering logic, then the RD BUFFER REQUEST signal is iteratively examined at step 75 until it indicates a request to dequeue a read command from the read command queue. According to another embodiment of the present invention, step 75 may be skipped because the RD BUFFER REQUEST signal is asserted any time the RD BUFFER EMPTY signal is asserted.

Detailed Description Text (68):
At step 77, the fenced read command is dequeued from the read command queue and at step 79 the synchronization value is dequeued from the head of the write command queue. As discussed above, steps 77 and 79 may be performed concurrently or in reverse order from that depicted in method 100. After the fenced read command and the synchronization value have been dequeued from their respective command queues, method execution loops back to step 55.

Detailed Description Text (69):

As shown in FIG. 13, steps 81, 83, 85, 87, 89 and 91 mirror steps 67, 69, 71, 73, 75, 77 and 79 described above, except that the roles of the read command queue and the write command queue are reversed. Also, while steps 55 and 57 of method 100 are described as determining whether a fenced read command or a fenced write command, respectively, is present at the head of a command queue, each of steps 55 and 57 must also include a determination of whether a synchronization value is present at the head of the command queue. This is because, according to one embodiment of the present invention, a synchronization value may not be dequeued from the head of one of the write and read command queues unless a corresponding fenced memory command is present at the head of the other of the write and read command queues. Additional steps (not shown), essentially mirroring the logic of steps 67, 69, 71, 73, 75, 77 and 79, must be executed in method 100 to ensure that a fenced memory access command is advanced to the head of a command queue upon determining that a synchronization value is present at the head of the other command queue. After both the synchronization value and the fenced memory access command have been advanced to the head of their respective command queues, and after memory access commands previously dequeued to the AGP command reordering logic have been ordered for execution in a fixed sequence, the synchronization value and the fenced memory access command may be dequeued.

CLAIMS:

5. A method comprising:

receiving a fence command in a device that implements command reordering;

queuing a first command that succeeds the fence command in either a read-command queue when the first command indicates a memory read operation or a write-command queue when the first command indicates a memory write operation;

queuing a synchronization value either in the read-command queue when the first command indicates the memory write operation or in the write-command queue when the first command indicates the memory read operation; and

delaying the first command from being dequeued until commands received prior to the fence command have been dequeued and ordered for execution in a fixed sequence.

6. The method of claim 5 further comprising:

delaying the first command from being dequeued until the synchronization value is advanced to a head of the read-command queue if the first command indicates the memory write operation; and

delaying the first command from being dequeued until the synchronization value is advanced to a head of the write-command queue if the first command indicates the memory read operation.

7. The method of claim 5 wherein receiving a fence command in a device that implements command reordering comprises receiving a fence command in a memory controller that implements command reordering.

9. In a memory controller having a first queue and a second queue wherein memory access commands available to be dequeued from one of the first and second queues may be dequeued irrespective of whether memory access commands are available to be dequeued from the other of the first and second queues, a method comprising the steps of:

receiving in succession a fence command and a first memory access command;

queuing in the first queue the first memory access command together with an indication that the first memory access command succeeded the fence command;

queuing a synchronization value in the second queue; and

maintaining the first memory access command in the first queue until the synchronization value is advanced to a head of the second queue.

10. The method of claim 9 wherein the first memory access command is a memory write command.

11. An apparatus comprising:

a fence flag that is set to a first state when a fence command is received in said apparatus and to a second state when an executable command is received in said apparatus;

a first plurality of storage locations to queue executable commands of a first type, each of said first plurality of storage locations including at least one fence bit to indicate that an executable command stored therein was received when said fence flag was in the first state; and

a second a plurality of storage locations to queue executable commands of a second type, each of said second plurality of storage locations including at least one fence bit to indicate that an executable command stored therein was received when said fence flag was in the first state.

Previous Doc Next Doc Go to Doc#

L10: Entry 1 of 9 File: USPT Nov 23, 2004

DOCUMENT-IDENTIFIER: US 6823336 B1
TITLE: Data storage system and method for uninterrupted read-only access to a consistent dataset by one host processor concurrent with read-write access by another host processor.

Brief Summary Text (5):
Transaction processing uses computer programming techniques that maintain database consistency under various conditions such as recovery from system failure or concurrent database access by multiple host processors. Database consistency is typically maintained by subdividing the application program of a host processor into a series of transactions. Each transaction includes a set of read-write instructions that change the database from one consistent state to another. The set of read-write instructions for each transaction is terminated by an instruction that specifies a transaction commit operation. During the execution of the transaction, the database may become inconsistent. For example, in an accounting application, a transaction may have the effect of transferring funds from a first account to a second account. The application program has a first read-write instruction that debits the first account by a certain amount, and a second read-write instruction that credits the second account by the same amount. Before and after the transaction, the database has consistent states, in which the total of the funds in two accounts is constant. In other words, the total of the funds in the two accounts at the beginning of the transaction is the same as the total at the end of the transaction. During the transaction, the database will have an inconsistent state, in which the total of the funds in the two accounts will not be the same as at the beginning or at the end of the transaction.

Brief Summary Text (6):
For concurrent database access by multiple host processors, it is conventional to permit only one host processor to have read-write access to the database, and to permit the other host processors to have read-only access to the database. For many applications, read-only access to the database must be restricted to consistent states of the database. For example, decision support systems analyze and evaluate data that is accumulated in the course of business transactions. In such a decision support system, one host processor may perform read-write transactions upon a database to produce a record of the business transactions, and any number of other host processors may perform read-only access to the database to analyze and evaluate the data in the database. The analysis and evaluation must be performed upon consistent states of the database. For an accounting application, for example, an evaluation of the total amount of funds in a set of accounts would be erroneous if the totals were not computed from consistent states of the database.

Brief Summary Text (7):
There are various ways of restricting read-only access to consistent states of a database in a multi-processor environment. A typical way is to restrict read-only access to a snapshot copy of the database. The snapshot copy of the database is updated at the end of each transaction, at the conclusion of the transaction commit operation. The snapshot copy can be maintained by a data storage system that provides read-write access to the database and concurrent read-only access to the snapshot copy of the database. The snapshot copy of the database can be updated very quickly at the conclusion of the transaction commit operation, so as to provide uninterrupted read-only access to consistent states of the database.

Brief Summary Text (8):
For some situations, there are difficulties in using a conventional snapshot copy facility for providing uninterrupted read-only access to consistent states of the database. For example, Raz et al., U.S. Pat. No. 5,852,715, incorporated herein by reference, discloses a multi-processor system in which a local data storage system provides read-write access to a database, and a remote data storage system provides read-only access to the database. A data communications

link connects the remote data storage system to the local data storage system. The local data storage system stores a local copy of the database. The local copy of the database is mirrored over the data communications link to a remote copy of the database in the remote data storage system. At the remote data storage system, a support copy is derived from the remote database. The remote data storage system provides read-only access to the support copy to implement decision support functions. Raz et al. U.S. Pat. No. 5,852,715 discloses two ways of updating the remote copy. Changes made to the local database could be recorded in the remote database on an ongoing basis, in which case the support copy would be a snapshot of the remote database. Alternatively the remote copy could be a snapshot of the local copy. In either case, one could use a conventional remote mirroring facility and a conventional snapshot facility. However, it would appear that the use of such conventional facilities would require at least three versions of the database (local copy, snapshot copy, and mirrored copy), together with overhead for maintaining both the snapshot copy and the mirrored copy on an ongoing basis for concurrent and uninterrupted read-only access.

Brief Summary Text (12):
In accordance with yet another aspect, the invention provides a data storage system including data storage, and a storage controller responsive to read and write commands for accessing specified data of a dataset in the data storage. Each set of write commands modifies the dataset from one consistent state to another. The storage controller is programmed to respond to each set of write commands by first operating upon revisions of each set of write commands in a write-selected phase and then operating upon the revisions of each set of write commands in a read-selected phase. The storage controller forms a directory of the revisions of each set of write commands in the write-selected phase. The storage controller accesses the directory of the revisions of each set of write commands in the read-selected phase. The storage controller performs the revisions of each set of write commands in the read-selected phase upon the dataset, and concurrently responds to the read commands on a priority basis by accessing the directory of the revisions of said each set of write commands in the read-selected phase to obtain specified data from the revisions of each set of write commands in the read-selected phase when the specified data are in the revisions of each set of write commands in the read-selected phase, and when the specified data are not in the revisions of each set of write commands in the read-selected phase, obtaining the specified data from the dataset.

Brief Summary Text (13):
In accordance with still another aspect, the invention provides a data storage system including data storage, and a storage controller responsive to read and write commands for accessing specified data of a dataset in the data storage. The storage controller is programmed to respond to transaction commit commands by alternately writing to a first volume of the data storage and to a second volume of the data storage sets of revisions made to the dataset by the write commands. Each set of revisions to the dataset includes revisions from a set of transactions defined by the transaction commit commands so that each set of revisions changes the dataset from one consistent state to another. In addition, the storage controller is programmed with a remote mirroring facility for mirroring the first and second volumes to corresponding volumes in a remote data storage system.

Brief Summary Text (14):
In accordance with a final aspect, the invention provides a program storage device containing a program for a storage controller of a data storage system. The program is executable by the storage controller for responding to read and write commands for accessing specified data of a dataset in data storage of the data storage system. Each set of write commands modifies the dataset from one consistent state to another. The program is executable by the storage controller for responding to each set of write commands by first operating upon revisions of each set of write commands in a write-selected phase and then operating upon the revisions of each set of write commands in a read-selected phase. The storage controller forms a directory of the revisions of each set of write commands in the write-selected phase, and the storage controller accesses the directory of the revisions of each set of write commands in the read-selected phase. The storage controller performs the revisions of each set of write commands in the read-selected phase upon the dataset, and concurrently responds to the read commands on a priority basis by accessing the directory of the revisions of each set of write commands in the read-selected phase to obtain specified data from the revisions of each set of write commands in the read-selected phase when the specified data are in the revisions of each set of write commands in the read-selected phase, and when the specified data are not in the revisions of each set of write commands in the read-selected phase, obtaining the specified data from the

dataset.

Drawing Description Text (5):
FIG. 4 is a flow chart showing how the secondary data storage system in FIG. 1 is programmed to respond to a write command received from the primary data storage system;

Drawing Description Text (6):
FIG. 5 is a flow chart showing how the secondary data storage system in FIG. 1 is programmed to respond to a read command received from the secondary host processor;

Drawing Description Text (9):
FIG. 8 is a block diagram of a preferred construction for the data processing system of FIG. 1, in which a pair of "delta volumes" are mirrored between a primary data storage system and a secondary data storage system in order to buffer transmission of write commands from the primary data storage system to the secondary data storage system;

Drawing Description Text (13):
FIG. 12 is a flow chart of programming in a delta volume facility of the primary data storage system of FIG. 8 for remote transmission of write commands to the secondary data storage system;

Drawing Description Text (14):
FIG. 13 is a block diagram of an alternative embodiment of the invention, in which the data storage systems are file servers, and the write commands include all file system access commands that modify the organization or content of a file system; and

Detailed Description Text (3):
With reference to FIG. 1, there is shown a data processing system in which a primary data storage system 20 servicing a primary host processor 21 is connected via a transmission link 22 to a secondary storage system 23 servicing a secondary host processor 24. The primary data storage system 20 includes a storage controller 25 controlling access to primary storage 26, and the secondary data storage system 23 has a storage controller 27 controlling access to secondary storage 28. The storage controller 25 is programmed, via a program storage device such as a floppy disk 29, with a remote mirroring facility 30, which transmits write commands from the primary host processor 21 over the link 22 to the storage controller 27 in the secondary storage system. The storage controller 27 receives the write commands and executes them to maintain, in the secondary storage 28, a copy of data that appears in the primary storage 26 of the primary data storage system. Further details of a suitable remote mirroring facility are disclosed in Ofek et al., U.S. Pat. No. 5,901,327 issued May 4, 1999, incorporated herein by reference.

Detailed Description Text (4):
In accordance with an aspect of the present invention, the storage controller 27 in the secondary data storage system is programmed with a concurrent access facility for providing the secondary host processor 24 uninterrupted read-only access to a consistent dataset in the secondary storage 28 concurrent with read-write access by the primary host processor. For example, the concurrent access facility 31 is loaded into the storage controller 27 from a program storage device such as a floppy disk 32. The concurrent access facility 31 is responsive to the write commands from the primary data storage system, and read-only access commands from the secondary processor 24. The concurrent access facility 31 is also responsive to transaction commit commands, which specify when the preceding write commands will create a consistent dataset in the secondary storage 28. The transaction commit commands originate from the primary host processor 21, and the storage controller 25 forwards at least some of these transaction commit commands over the link 22 to the storage controller 27.

Detailed Description Text (6):
To provide the secondary host processor with uninterrupted read-only access to a consistent dataset, the switches 45 and 46 are toggled in response to receipt of a transaction commit command received over the link 22 from the primary data storage system. Moreover, the switches 45 and 46 are not toggled unless all of the revisions in the read-selected storage "A" or "B" of dataset revisions have been transferred to the dataset secondary storage 42, and unless all of the updates since the last transaction commit command have actually been written from the link 22 into the write-selected storage "A" or "B" of dataset revisions. (For the switch

positions in FIG. 2, the storage "A" of dataset revisions 43 is write selected, and the storage "B" of dataset revisions is read-selected.) Therefore, the combination of the dataset revisions in the read-selected storage "A" or "B" of dataset revisions with the dataset in the dataset secondary storage represents a consistent dataset. Just after the switches 45 and 46 are toggled, the secondary data storage system begins a background process of reading dataset revisions from the read-selected storage "A" or "B" of dataset revisions, and writing the updates into the dataset secondary storage. Moreover, at any time the secondary host processor 24 may read any dataset revisions from the read-selected storage "A" or "B" of dataset revisions. If a dataset revision is not found in the read-selected storage "A" or "B" of dataset revisions for satisfying a read command from the secondary host processor 24, then read data is fetched from the dataset secondary storage 42.

Detailed Description Text (7):
One advantage to the present invention is that the concurrent access facility 31 can provide the secondary host processor with substantially uninterrupted and concurrent read-only access to a consistent dataset regardless of the rate at which the dataset secondary storage 42 is updated to a consistent state by the completion of integration of a set of revisions into the dataset secondary storage. Therefore, the dataset in the dataset secondary storage 42 can be updated at a relatively low rate, and the storage controller 25 of the primary data storage system 20 can send transaction commit commands to the storage controller 27 of the secondary data storage system 23 at a much lower rate than the rate at which the storage controller 25 receives transaction commit commands from the primary host processor 21. Moreover, the transaction commit commands can be encoded in the write commands sent over the link. For example, the write commands can write alternate sets of revisions to alternate dataset revision storage, as will be described below with respect to FIG. 9. In such a case, the storage controller 27 in the secondary data storage system 23 can regenerate the transaction commit commands by detecting that the addresses of the write commands have switched from one area of dataset revision storage to the other. Moreover, each write command can be tagged with a corresponding sequence number so that the storage controller 27 in the secondary data storage system 23 can verify that a complete set of write commands has been received prior to the switch of the write command addresses from one area of the dataset revision storage to the other.

Detailed Description Text (8):
FIG. 3 is a block diagram showing control flow through the secondary data storage system of FIG. 1. Upon receipt of a write command (from the link 22 in FIGS. 1 and 2), the secondary data storage system accesses a directory 47 or 48 for the write-selected storage "A" or "B" of dataset revisions. The directory is accessed to determine whether or not the write command is accessing the same data item or data storage location as an update existing in the write-selected storage "A" or "B" of dataset revisions. If so, then the directory provides the location of the update in the write-selected storage "A" or "B" of dataset revisions, and the write command is executed upon that pre-existing update. If not, then storage is allocated in the write-selected storage "A" or "B" of dataset revisions for the update of the write command, the update of the write command is written into the allocated storage, and the directory 47 or 48 of the write-selected storage "A" or "B" of dataset revisions is updated to associate the allocated storage for the storage location or data item accessed by the write command.

Detailed Description Text (10):
FIG. 4 is a flow chart showing how the secondary data storage system in FIG. 1 is programmed to respond to a write command received from the primary data storage system. The write command specifies an address of a data item or storage location, and data to be written to the data item or storage location. In the first step 61, the storage controller accesses the write-selected directory "A" or "B" of dataset revisions (47 or 48) for the address specified by the write command. Next, in step 62, execution branches depending on whether or not the address is in the directory. If not, then in step 63, the storage controller allocates storage for the write data in the write-selected storage "A" or "B" of dataset revisions (43 or 44). Then in step 64, the storage controller writes the data to the allocated storage. Then in step 65, the storage controller creates a new directory entry (in the write-selected directory "A" or "B" of dataset revisions 47 or 48) associating the write address with the allocated storage. Then in step 66, the storage controller returns an acknowledgement over the link to the primary storage system, and the task is finished.

Detailed Description Text (11):

In step 62, if the write address is in the directory, then execution branches to step 67. In step 67, the storage controller writes the data of the write command to the associated address in the write-selected storage "A" or "B" of dataset revisions (43 or 44). Execution continues from step 67 to step 66 to return an acknowledgement to the primary storage system, and the task is finished.

Detailed Description Text (12):
FIG. 5 is a flow chart showing how the storage controller of the secondary data storage system in FIG. 1 is programmed to respond to a read command received from the secondary host processor. The read command specifies an address of a data item or storage location. In a first step 71, the storage controller accesses the read-selected directory "A" or "B" of dataset revisions (47 or 48). Then in step 72, execution branches depending on whether the address in the read command is found in the directory. If so, then execution branches from step 72 to step 73. In step 73, the storage controller reads data from the read-selected storage "A" or "B" of dataset revisions. Execution continues from step 73 to step 74, to return the data to the secondary host processor, and then the task is finished.

Detailed Description Text (13):
If in step 72 the read address is not in the directory accessed in step 71, then execution continues from step 72 to step 75. In step 75, the storage controller accesses the dataset directory (48 in FIG. 3). Then in step 76, execution branches depending on whether the address of the read command is in the dataset directory. If not, execution continues to step 77, to return an error code to the secondary host processor, and then the task is finished. Otherwise, if the address of the read command is found in the dataset directory, execution branches from step 76 to step 78. In step 78, the storage controller reads data from the dataset secondary storage (42 in FIG. 3). Execution continues from step 78 to step 74, to return the data to the secondary host processor, and the task is finished.

Detailed Description Text (14):
FIG. 6 is a flowchart showing how the storage controller of the secondary data storage system in FIG. 1 is programmed to respond to a transaction commit command from the primary data storage system. In a first step 81, the storage controller checks whether or not the background task of FIG. 7 is done with integration of the dataset into the dataset secondary storage. For example, this background task is done when the read-selected directory "A" or "B" of dataset revisions is empty. If not, then in step 82, the storage controller returns a flow control signal to the primary data storage system, because subsequent write commands from the link should not be placed in the storage "A" or "B" of dataset revisions until completion of the integration of the dataset revisions into secondary storage. Any such subsequent write commands could be placed in a temporary buffer until completion of the integration of the dataset revisions into the secondary storage, and a preferred buffering technique will be described below with reference to FIGS. 8 to 11. Execution continues from step 82 to step 83. In step 83, the task of FIG. 6 is suspended for a time to permit the background task to continue with integration of the dataset into secondary storage, and then the task is resumed. After step 83, execution loops back to step 81. Once the dataset has been integrated into secondary storage, execution continues from step 81 to step 84.

Detailed Description Text (15):
In step 84, the switches (45 and 46 in FIGS. 2 and 3) are toggled. This is done by complementing a logical variable or flag, which indicates what storage of dataset revision is selected for read and write operations. For example, when the flag has a logical value of 0, the storage "A" of dataset revisions 43 is read-selected, and the storage "B" of dataset revisions 44 is write-selected. When the flag has a logical value of 1, the storage "A" of dataset revisions 43 is write-selected, and the storage "B" of dataset revisions is read-selected. Next, in step 85, the storage controller initiates the background task of integrating dataset revisions from the read-selected storage "A" or "B" of dataset revisions into the dataset secondary storage. Then, in step 86, the storage controller returns an acknowledgement of the transaction commit command to the primary data storage system, and the task of FIG. 6 is done.

Detailed Description Text (16):
FIG. 7 is a flow chart showing how the storage controller of the secondary data storage system in FIG. 1 is programmed to perform a background task of integrating revisions into the dataset secondary storage. In a first step 91, the first dataset revision is obtained from the read-

selected "A" or "B" dataset revision storage (43 or 44 in FIG. 3). Next, in step 92, the storage controller searches the dataset directory (49 in FIG. 3) for the write address of the dataset revision. Then, in step 93, execution branches depending on whether the write address is found in the directory. If not, execution continues from step 93 to step 94. In step 94, the storage controller stores the revision in the dataset secondary storage (42 in FIG. 3), and the storage controller updates the dataset directory (49 in FIG. 3). Execution continues from step 94 to step 96.

Detailed Description Text (18):
In step 96, the storage controller de-allocates storage of the dataset revision from the read-selected "A" or "B" dataset revision storage (43 or 44 in FIG. 3). Execution continues from step 96 to step 97. In step 97, the task is finished if the dataset revision storage is found to be empty. Otherwise, execution continues from step 97 to step 98. In step 98, the task is suspended to permit any higher priority tasks to begin, and once the higher priority tasks are completed, the background task is resumed. Execution then continues to step 99. In step 99, the storage controller obtains the next dataset revision from the read-selected "A" or "B" dataset revision storage. Execution loops back to step 92 from step 99, in order to integrate all of the revisions from the read-selected "A" or "B" dataset revision storage into the dataset secondary storage.

Detailed Description Text (19):
The above description with respect to FIGS. 1 to 5 has not been limited to any particular form of dataset structure or directory structure. For example, the dataset revisions could operate upon direct mapped, numerically addressed storage, or they could operate upon dynamically allocated, symbolically addressed storage. For example, FIG. 8 is a block diagram of one preferred construction for a data processing system in which the write commands for the dataset revisions access direct mapped, numerically addressed storage. The data processing system includes a primary data storage system 110, a data mover computer 111, a primary host processor 112, a secondary data storage system 113, a data mover computer 114, and a secondary host processor 115. The data mover computer 111 includes a file system 116 that translates file system read and write commands from the primary host processor 112 to logical block read and write commands to the primary data storage system. Therefore, the combination of the data mover computer 111 and the primary data storage system 110 functions as a file server. Further details regarding the programming of the data mover computer 111 and the file system 116 are disclosed in Vahalia et al., U.S. Pat. No. 5,893,140, issued Apr. 6, 1999, and entitled "File Server Having A File System Cache And Protocol For Truly Safe Asynchronous Writes," incorporated herein by reference. In a similar fashion, the combination of the secondary data storage system 113 and the data mover computer 114 also functions as a file server.

Detailed Description Text (20):
The primary data storage system has primary storage 118, and a storage controller 119. The storage controller includes a semiconductor random access cache memory 120, a host adapter 121 interfacing the data mover computer 111 to the cache memory, disk adapters 122, 123 interfacing the cache memory to the primary storage 118, and a remote mirroring facility 124 for interfacing the cache memory 120 to dual redundant data transmission links 125, 126 interconnecting the primary data storage system 110 to the secondary data storage system 113. The remote mirroring facility is constructed and operates as described in the above-cited Ofek et al., U.S. Pat. No. 5,901,327 issued May 4, 1999. This remote mirroring facility mirrors file system storage 141 in the primary storage 118. However, the file system storage 141 is mirrored by mirroring delta volume storage 143 that is used to buffer the updates to file system storage 141 of the primary storage 118. The host adapter 121 is programmed with a "delta volume facility" 127 that loads the updates into the delta volume storage 143 of the primary storage 118. The remote mirroring facility transmits the updates over the dual redundant links 125, 126 to mirrored delta volume storage 144 in secondary storage 128 in the secondary data storage system 113, as further shown and described below with reference to FIGS. 9 to 12.

Detailed Description Text (22):
The secondary data storage system 113 also includes a storage controller 129. The storage controller 129 includes a semiconductor cache memory 130, a host adapter 131, disk adapters 132 and 133, and a remote mirroring facility 134. The host adapter 131 is programmed with a concurrent access facility 135 that is similar to the concurrent access facility (31 in FIG. 1) described above with respect to FIGS. 1 to 7, except that the concurrent access facility 135 obtains updates from the mirrored delta volume storage 144 in the secondary storage 128 (as

further described below with reference to FIGS. 9 to 11) instead of directly from the primary data storage system.

Detailed Description Text (28):
FIG. 12 is a flowchart of programming in a delta volume facility of the primary data storage system of FIG. 8 for remote transmission of write commands to the secondary data storage system. In a first step 161, the storage controller of the primary data storage system clears the sequence number (SEQNO). The sequence number is used to map the current delta chunk into either the "A" delta volume or the "B" delta volume. For example, if the sequence number is even, then the current delta chunk is in the "A" delta volume, and if the sequence number is odd, then the current delta chunk is in the "B" delta volume. For the case of four delta chunks per delta volume, for example, the position of the delta chunk in the corresponding delta volume is computed by an integer division by two (i.e., a right shift by one bit position), and then masking off the two least significant bits (i.e., the remainder of an integer division by four).

Detailed Description Text (29):
Next, in step 162, the storage controller clears a variable indicating the delta set size (DSS). Then in step 163, the storage controller clears a timer. The timer is a variable that is periodically incremented. The timer is used to limit the frequency at which transaction commit commands are forwarded from the primary data storage system to the secondary storage system unless the transaction commit commands need to be transmitted at a higher rate to prevent the size of the delta sets from exceeding the size of the delta chunk.

Detailed Description Text (30):
In step 164, execution continues to step 165 if the storage controller receives a write command from the primary host processor. In step 165, the storage controller places the write command in the current delta chunk. This involves writing a number of data blocks to the delta volume selected by the sequence number (SEQNO), beginning at an offset computed from the sequence number and the current delta set size (DSS). Then in step 166, the storage controller increments the delta set size (DSS) by the number of blocks written to the delta chunk. In step 167, the storage controller compares the delta set size to a maximum size (DSM) to check whether a delta chunk overflow error has occurred. If so, then execution branches to an error handler 168. Otherwise, execution continues to step 169. In step 169, execution continues to step 170 unless a transaction commit command is received from the primary host processor. If not, execution continues to step 170, to temporarily suspend, and then resume, the delta volume facility task of FIG. 12. Otherwise, if a transaction commit command is received, execution continues to step 171. It should be noted that once step 171 has been reached, the data mover computer (111 in FIG. 8) has already flushed any and all file system updates preceding the transaction commit command from any of its local buffer storage to the primary data storage system. Write operations by the primary host processor subsequent to the transaction commit command are temporarily suspended until this flushing is finished. Therefore, once step 171 has been reached, the updates in the delta set of the current delta chunk represent a change of the file system from one consistent state to another consistent state.

Detailed Description Text (31):
In step 171, the storage controller compares the delta set size (DSS) to a threshold size (THS) that is a predetermined fraction (such as one-half) of the maximum size (DSM) for a delta chunk. If the delta set size (DSS) is not greater than this threshold, then execution continues to step 172. In step 172, the timer is compared to a predetermined threshold (THT) representing the minimum update interval for the file system secondary storage unless a smaller update interval is needed to prevent the size of the delta set from exceeding the size of the delta chunk. The minimum update interval (THT) should depend on the particular application. A value of 5 minutes for THT would be acceptable for many applications. If the timer is not greater than the threshold (THT), then execution loops back to step 170. Otherwise, execution continues to step 173. Execution also branches from step 171 to step 173 if the delta set size (DSS) is greater than the threshold size (THS).

Detailed Description Text (32):
In step 173, the storage controller writes the delta set size (DSS) and the sequence number (SEQNO) into an attributes block of the delta chunk (e.g., the trailer 199 in FIG. 10.) The updating of the sequence number in the delta chunk validates the delta set in the delta chunk. Then, step 174, the storage controller flushes the current delta volume to ensure that all

updates in the delta set of the current delta chunk will be transmitted to the secondary data storage system, and then sends a transaction commit command to the secondary data storage system. The secondary data storage system should have received all of the updates in the delta set of the current delta chunk before receipt of the transaction commit command. For example, the remote data mirroring facility can be operated in an asynchronous or semi-synchronous mode for the current delta volume until step 174, and switched in step 174 to a synchronous mode to synchronize the current delta volume in the primary data storage system with its mirrored volume in the secondary data storage system, and then the transaction commit command can be sent once the remote mirroring facility indicates that synchronization has been achieved for the current delta volume. In step 175, the storage controller increments the sequence number (SEQNO). In step 176, the storage controller temporarily suspends the delta volume facility task of FIG. 12, and later resumes the task. Execution then loops back from step 176 to step 162.

Detailed Description Text (36):
FIG. 13 is a block diagram of an alternative embodiment of the invention, in which the data storage systems are file servers, and the write commands include all file system access commands that modify the organization or content of a file system. The data processing system in FIG. 13 includes a conventional primary file server coupled to a primary host processor 182 for read-write access of the primary host processor to a file system stored in the file server. The conventional file server includes a storage controller 183 and primary storage 184. The storage controller 183 includes facilities for file access protocols 191, a virtual file system 192, a physical file system 193, a buffer cache 194, and logical-to-physical mapping 195. Further details regarding such a conventional file server are found in the above-cited Vahalia et al., U.S. Pat. No. 5,893,140, issued Apr. 6, 1999.

Detailed Description Text (37):
The data processing system in FIG. 13 also includes a secondary file server 185 coupled to the primary host processor 182 to receive copies of at least the write access commands sent from primary host processor to the primary file server. The secondary file server has a storage controller 187 and secondary storage 188. The storage controller 187 includes facilities for file access protocols 201, a virtual file system 202, a physical file system 203, a buffer cache 204, and logical-to-physical mapping 205. To this extent the secondary file server is similar to the primary file server.

Detailed Description Text (38):
In the data processing system of FIG. 13, the primary host processor 182 has a remote mirroring facility 186 for ensuring that all such write access commands are copied to the secondary file server 185. (This remote mirroring facility 186 could be located in the primary file server 181 instead of in the primary host processor.) The remote mirroring facility 186 also ensures that the primary host processor will receive acknowledgement of completion of all preceding write commands from an application 199 from both the primary file server 181 and the secondary file server 185 before the primary host processor will return to the application an acknowledgement of completion of a transaction commit command from the application 199. (This is known in the remote mirroring art as a synchronous mode of operation, and alternatively the remote mirroring facility 186 could operate in an asynchronous mode or a semi-synchronous mode.) The secondary file server 185 therefore stores a copy of the file system that is stored in the primary file server 181. Moreover, a secondary host processor 189 is coupled to the secondary file server 185 for read-only access of the secondary host processor to the copy of the file system that is stored in the secondary storage.

Other Reference Publication (2):
King and Halim, "Management of a Remote Backup Copy for Disaster Recovery," ACM Transactions on Database Systems, vol. 16, No. 2, Jun. 1991, pp. 338-368.

Other Reference Publication (6):
Gorelik et al., "Sybase Replication Server," International Conference on Management of Data and Symposium on Principles of Database Management 1994 SIGMOD, pp. 469 (also submitted is a two-page portal.acm.org abstract).

Other Reference Publication (8):
Disaster Recovery Guidelines for using HP SureStore E XP256, Continous Access XP with Oracle Databases Rev 1.03, Hewlett-Packard Company, Palo Alto, CA, May 2000, pp. 1-28.

Other Reference Publication (10):
VERITAS Volume Replication and Oracle Databases, A Solutions White Paper, Veritas Software Corporation, Mountain View, CA, May 29, 2000, pp. 12-31.

CLAIMS:

20. A data storage system comprising, in combination, data storage, and a storage controller responsive to read and write commands for accessing specified data of a dataset in the data storage, wherein each set of write commands modifies the dataset from one consistent state to another, and the storage controller is programmed to respond to each set of write commands by first operating upon revisions of said each set of write commands in a write-selected phase and then operating upon the revisions of said each set of write commands in a read-selected phase, the storage controller building up a directory of the revisions of said each set of write commands in the write-selected phase, and the storage controller accessing the directory of the revisions of said each set of write commands in the read-selected phase, the storage controller performing the revisions of said each set of write commands in the read-selected phase upon the dataset, and concurrently responding to the read commands on a priority basis by accessing the directory of the revisions of said each set of write commands in the read-selected phase to obtain specified data from the revisions of said each set of write commands in the read-selected phase when the specified data are in the revisions of said each set of write commands in the read-selected phase, and when the specified data are not in the revisions of said each set of write commands in the read-selected phase, obtaining the specified data from the dataset.

21. The data storage system as claimed in claim 20, wherein the data storage system further includes a first buffer and a second buffer, and the storage controller is programmed to place the revisions of said each set of write commands alternately in either the first buffer or the second buffer and to select one of the buffers to access the revisions for one of the sets of write commands currently in the read-selected phase and to select the other of the buffers to access one of the sets of write commands currently in the write-selected phase, and to switch between the buffers to switch from the read-selected phase to the write-selected phase for the revisions of each of the sets of the write commands.

22. The data storage system as claimed in claim 21, wherein the storage controller is programmed to switch between the buffers in response to a transaction commit command.

23. The data storage system as claimed in claim 20, wherein the data storage system is a file server, the dataset is a file system, the read commands are file access commands, and the write commands are file access commands.

24. The data storage system as claimed in claim 23, wherein the directory of the revisions of said each set of write commands is a hierarchical data structure including at least a root directory, a subdirectory, and at least one file entry in the subdirectory.

25. A data storage system comprising, in combination: data storage, and a storage controller responsive to read and write commands for accessing specified data of a dataset in the data storage, the write commands specifying sets of revisions to be made to the dataset, wherein the storage controller is programmed to respond to transaction commit commands by alternately writing the sets of revisions to a first volume of the data storage and to a second volume of the data storage, each set of revisions to the dataset including revisions from a set of transactions defined by the transaction commit commands so that each set of revisions changes the dataset from one consistent state to another, and the storage controller is programmed with a remote mirroring facility for mirroring the first and second volumes to corresponding volumes in a remote data storage system.

26. The data storage system as claimed in claim 25, wherein the storage controller is programmed to respond to a transaction commit command by flushing a current set of revisions to one of the first or second volumes, the flushing operation including the remote mirroring facility transmitting the current set of revisions from said one of the first or second volumes to the remote data storage system, and then transmitting a transaction commit command to the remote data storage system.

27. The data storage system as claimed in claim 26, wherein the storage controller is programmed to select a number of revisions to be included in each set of revisions in order to obtain a limited frequency of transmission of transaction commit commands to the remote data storage system.

28. The data storage system as claimed in claim 27, wherein the storage controller is programmed to permit the frequency of transmission of transaction commit commands to the remote data storage system to exceed the limited frequency when required to avoid overflow of the first and second volumes.

29. The data storage system as claimed in claim 25, wherein the storage controller is programmed to assign a unique sequence number to each set of revisions and to store the sequence number with each set of revisions when each set of revisions is stored in one of the storage volumes.

30. The data storage system as claimed in claim 29, wherein each of said volumes has a plurality of regions, and the storage controller is programmed to map each sequence number to a respective one of the regions for storage of the set of revisions to which said each sequence number is assigned.

31. A program storage device comprising: a program for a storage controller of a data storage system, the program being executable by the storage controller for responding to read and write commands for accessing specified data of a dataset in data storage of the data storage system, wherein each set of write commands modifies the dataset from one consistent state to another, and the program is executable by the storage controller for responding to each set of write commands by first operating upon revisions of said each set of write commands in a write-selected phase and then operating upon the revisions of said each set of write commands in a read-selected phase, the storage controller building up a directory of the revisions of said each set of write commands in the write-selected phase, and the storage controller accessing the directory of the revisions of said each set of write commands in the read-selected phase, the storage controller performing the revisions of said each set of write commands in the read-selected phase upon the dataset, and concurrently responding to the read commands on a priority basis by accessing the directory of the revisions of said each set of write commands in the read-selected phase to obtain specified data from the revisions of said each set of write commands in the read-selected phase when the specified data are in the revisions of said each set of write commands in the read-selected phase, and when the specified data are not in the revisions of said each set of write commands in the read-selected phase, obtaining the specified data from the dataset.

32. The program storage device as claimed in claim 31, wherein the data storage system further includes a first buffer and a second buffer, and the program is executable by the storage controller for placing the revisions of said each set of write commands alternately in either the first buffer or the second buffer and to select one of the buffers to access the revisions for one of the sets of write commands currently in the read-selected phase and to select the other of the buffers to access one of the sets of write commands currently in the write-selected phase, and to switch between the buffers to switch from the read-selected phase to the write-selected phase for the revisions of each of the sets of the write commands.

33. The program storage device as claimed in claim 32, wherein the program is executable by the storage controller for switching between the buffers in response to a transaction commit command.

Previous Doc Next Doc Go to Doc#